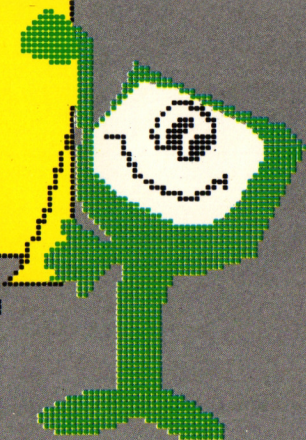
VIDEO BASIC

20 VIDEOLEZIONI DI BASIC
PER IMPARARE CON L'MSX

GRUPPO EDITORIALE JACKSON

11

MSX

Per tutti i sistemi MSX

VIDEOBASIC MSX
Pubblicazione quattordicinale
edita dal Gruppo Editoriale Jackson
Direttore Responsabile:
Giampietro Zanga
Direttore e Coordinatore
Editoriale: Roberto Pancaldi
Autore: Softidea -
Via Indipendenza 88-90 - Como
Redazione software:
Michele Casartelli
Francesco Franceschini
Progetto grafico:
Studio Nuovidea - via Longhi, 16 - Milano
Impaginazione:
Moreno Confalone
Illustrazioni:
Cinzia Ferrari, Silvano Scolari
Fotografie:
Marcello Longhini
Distribuzione: SODIP
Via Zuretti, 12 - Milano
Fotocomposizione: Lineacomp S.r.l.
Via Rosellini, 12 - Milano
Stampa: Grafika '78
Via Trieste, 20 - Pioltello (MI)
Direzione e Redazione:
Via Rosellini, 12 - 20124 Milano
Tel. 02/6880951/5

Autorizzazione alla pubblicazione Tribunale di Milano n° 422 del 22-9-1984
Spedizione in abbonamento postale Gruppo II/70 (autorizzazione della Direzione Provinciale delle PPTT di Milano).
Prezzo del fascicolo L. 8.000
Abbonamento comprensivo di 5 raccoglitori L. 165.000
I versamenti vanno indirizzati a: Gruppo Editoriale Jackson S.p.A. - Via Rosellini, 12 20124 Milano, mediante emissione di assegno bancario o cartolina vaglia oppure utilizzando il c.c.p. n° 11666203.
I numeri arretrati possono essere richiesti direttamente all'editore inviando L. 10.000 cdu. mediante assegno bancario o vaglia postale o francobolli.
Non vengono effettuate spedizioni contrassegno.

GRUPPO EDITORIALE
JACKSON
DIVISIONE GRANDI OPERE

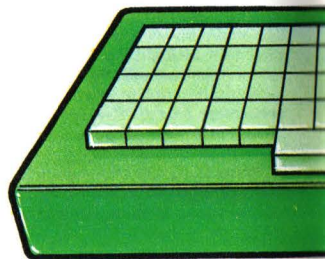
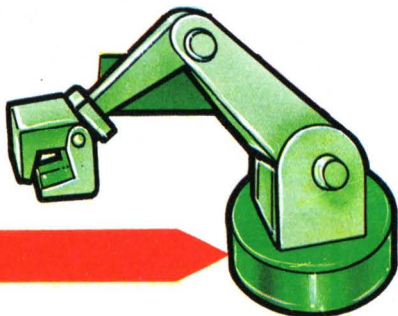
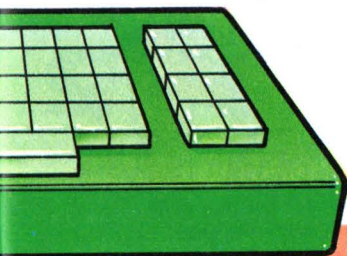
SOMMARIO

Introduzione

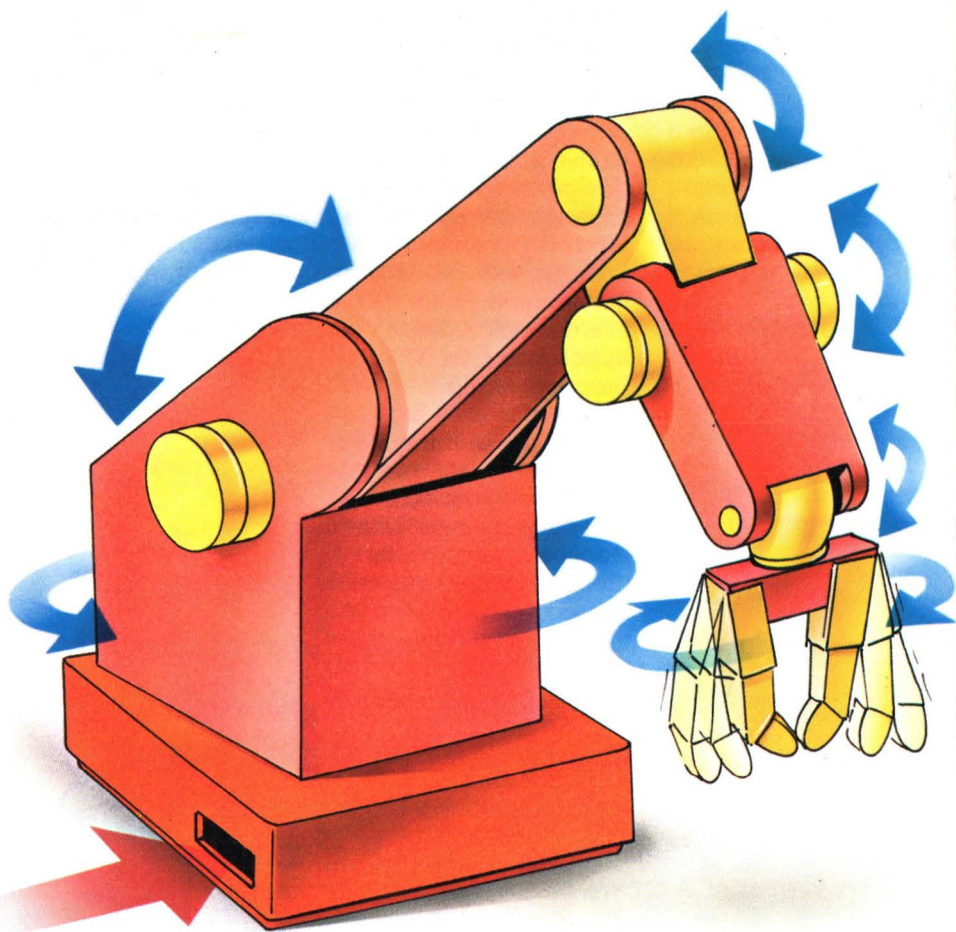
In questa lezione vedremo innanzitutto come funzionano, come apprendono e come lavorano i robot, accennando anche alle affascinanti prospettive e possibilità che questi dispositivi promettono di aprire nella vita di tutti i giorni.
Scopriremo quindi in quale modo sia possibile inserire permanentemente in un programma informazioni utili o ricorrenti, utilizzando le istruzioni READ, DATA e RESTORE.
Per finire, vedremo un esempio sulla tecnica da adottare per personalizzare le visualizzazioni in uscita, imparando come definire e memorizzare nuovi caratteri detti SPRITE, all'interno del nostro computer.

Cos'è un robot

Sin da prima dell'avvento dei calcolatori elettronici, una delle maggiori aspirazioni dell'uomo fu quella di riuscire ad inventare e costruire nuove macchine, che potessero sostituirlo nei lavori più faticosi, pericolosi o ripetitivi.

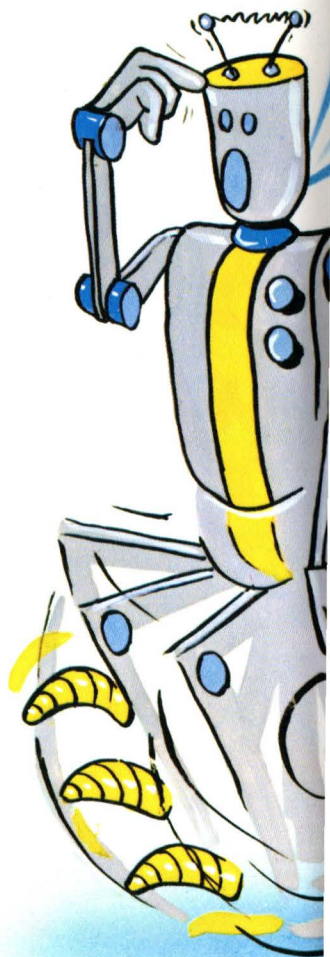
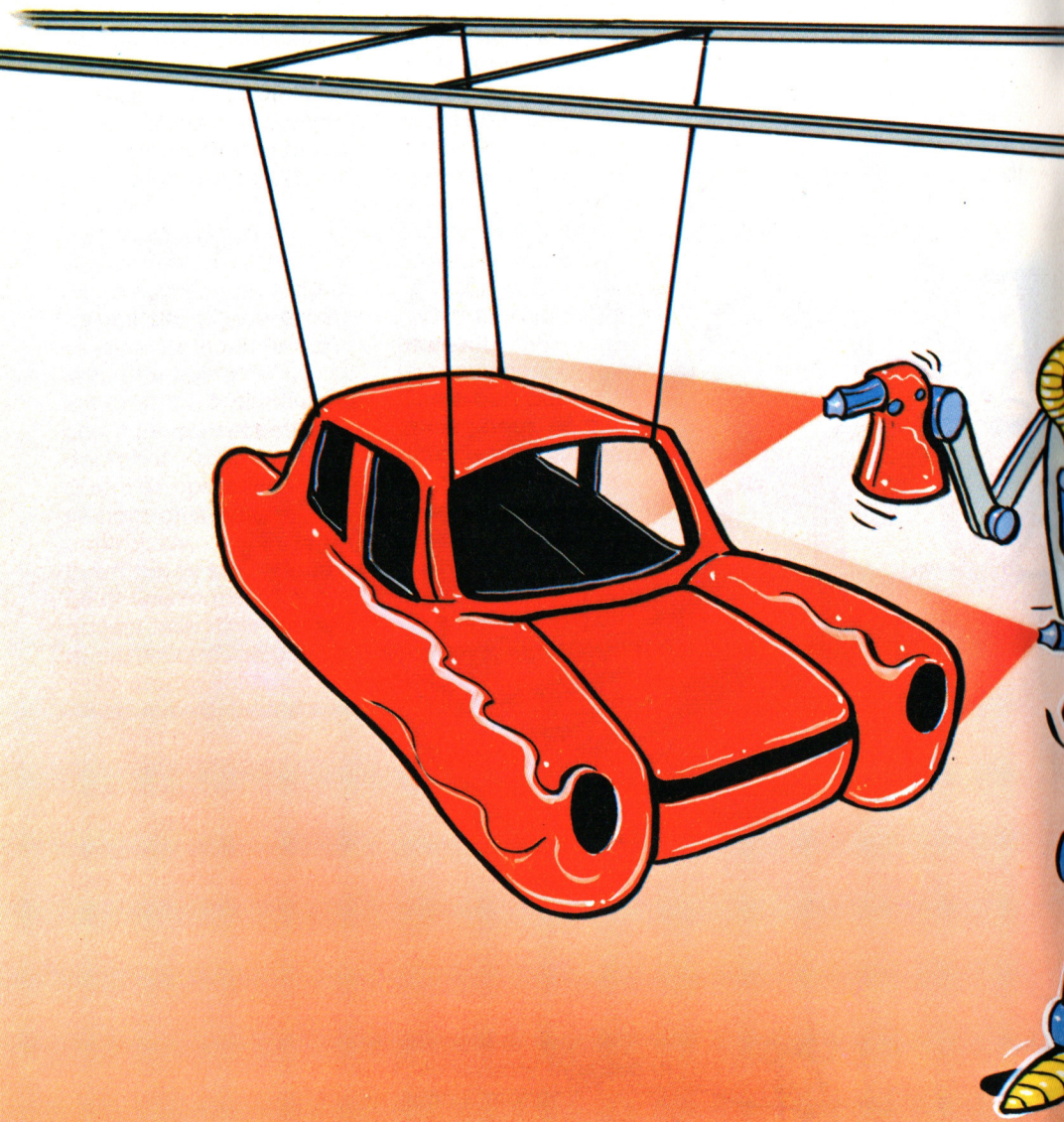
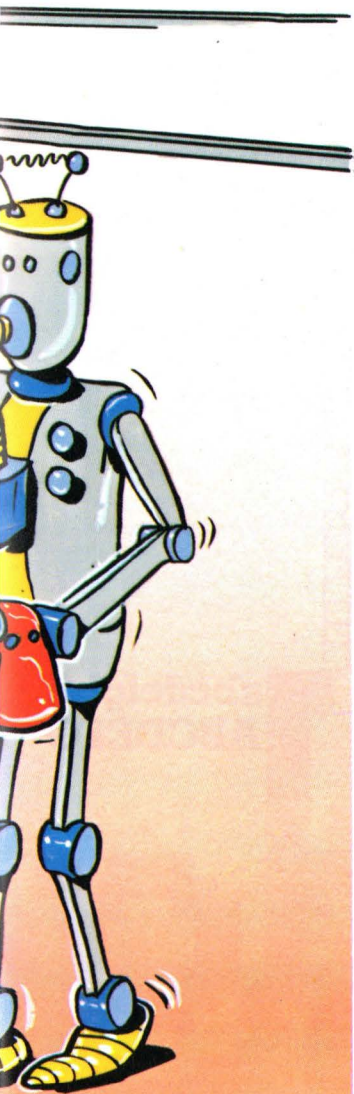
La storia ce lo insegna: ciascun gradino della scala del progresso è stato (ed è tuttora) normalmente caratterizzato da una grande e fondamentale invenzione o scoperta (come il fuoco, il ferro, il vapore o l'elettricità), che - inizialmente realizzata per risolvere determinati problemi - una volta applicata e sviluppata ci ha procurato (e ci procura) nuovi e magari inaspettati progressi e perfezionamenti.
Il calcolatore elettronico, per quanto non possa essere considerato una invenzione nel senso stretto della parola, non sfugge a questa regola. Nato per "calcolare", il computer è diventato la base per una infinità di nuove applicazioni, che su di esso si basano e dal quale dipendono strettamente. Tra queste la robotica, cioè la scienza che si occupa dell'automatizzazione del lavoro attraverso macchine controllate da computer.
Robot è un termine che nella mente di molte persone suscita tuttora parecchi dubbi e perplessità: il ricordo di uomini metallici e di macchine parlanti, tanto cari alla fantascienza più commerciale, ne rendono forse difficile l'esatta comprensione.
Per prima cosa cerchiamo allora di spiegarne il significato.
Un robot non è altro che una macchina automatica, la quale sotto il controllo di un computer, svolge un lavoro prestabilito: ad esempio verniciare un'automobile, avvitare dei bulloni o saldare delle lamiere.
Il grosso vantaggio di un robot rispetto ad una normale macchina automatica è che, essendo collegato ad un computer, può essere programmato - con modifiche di solito non

molto dispendiose - per svolgere lavori diversi (non troppo diversi, però!).
Alla programmabilità si aggiunge un altro pregio, grazie ai progressi della microelettronica i robot vengono infatti dotati di particolari dispositivi, chiamati sensori, che - per quanto ancora ben lontani dai sensi dell'uomo - permettono ad un robot di riconoscere il verificarsi di certe situazioni, particolari od anomale. Facciamo un esempio. Supponiamo di dover avvitare con un bullone due pezzi di metallo appositamente predisposti. Se utilizziamo per questa operazione un robot, possiamo metterci al riparo da ipotetiche evenienze, dovute al cattivo allineamento dei due pezzi, al difettoso fissaggio del bullone od all'errata dimensione di uno qualsiasi dei componenti. Fornendo alla macchina degli appositi sensori e programmando opportunamente il calcolatore di controllo è possibile arrestare la macchina al verificarsi di uno di questi casi. Una macchina normale, in presenza di uno di questi casi anomali, proseguirebbe imperterrita nel proprio lavoro, provocando magari la messa in produzione di un pezzo difettoso o, peggio ancora, la rottura della macchina stessa.
L'introduzione del robot inserisce quindi nella catena di montaggio un elemento di controllo e di verifica, oltre che di produzione. Molti robot possono inoltre svolgere lavori che per le persone sarebbero pericolosi o sgradevoli, come misurare la radioattività in una centrale nucleare, disinnescare una bomba o operare in atmosfere inquinate.

HARDWARE

Come insegnare ad un robot

La maggior parte dei robot esistenti lavora ed opera in fabbriche in cui tutto è organizzato intorno a loro: di solito debbono prelevare dei pezzi da nastri trasportatori, lavorarli seguendo una certa successione di operazioni elementari e depositarli infine su un altro nastro trasportatore, che provvede ad avviarli verso altre lavorazioni. Le prestazioni di un robot dipendono

comunque da molti fattori, non ultimo quello economico; esistono quindi (e sono la maggioranza, visto il loro minor costo e la relativa semplicità) robot, costruiti per un uso abbastanza generico, che di volta in volta vengono adattati per svolgere altre mansioni più specifiche.
I robot più comuni sono quelli cosiddetti "a braccio": la loro meccanica è infatti costruita prendendo a modello le braccia dell'uomo. Tale disposizione permette - proprio come per le braccia vere - una notevolissima libertà e capacità di movimenti, consentendo alle "mani" (che normalmente sono pinze comandate da motori elettrici) di operare in posizioni altrimenti inaccessibili.
Attualmente si sta anche studiando la maniera di dotare queste "mani" di una sorta di senso del tatto, consentendo così una presa adeguata al peso ed alla solidità dell'oggetto da sollevare (finora nessun robot riesce a distinguere un pezzo di ferro a forma d'uovo da un uovo vero, con risultati facilmente intuibili).
Progettare, costruire ed insegnare ad un robot è pertanto un'operazione per niente semplice: occorre innanzitutto analizzare e suddividere l'azione che il robot dovrà svolgere in tutti i possibili ed immaginabili passaggi elementari (ti ricordi gli algoritmi? ...), classificandone così le diverse necessità di movimento. In base a questi movimenti - ed in funzione dello sforzo necessario per compiere ciascuna azione - si deciderà il numero e la posizione dei bracci meccanici, dei giunti (cioè degli snodi tra i singoli bracci) e dei relativi motori.
Per eseguire il proprio lavoro il robot dovrà inoltre essere provvisto di sensori di vario tipo (fotoelettrici, elettronici o meccanici), appropriati al genere di lavorazione richiesto. Attraverso una o più interfacce questi sensori dovranno quindi inviare le informazioni codificate all'elaboratore di controllo, addetto alla direzione ed al coordinamento dei vari movimenti, che provvederà così a comandare l'esecuzione, la modifica o l'arresto delle operazioni previste in origine.
Il computer di controllo andrà pertanto programmato in modo adeguato a queste necessità, tenendo in debito conto tutte le possibilità di imprevisti

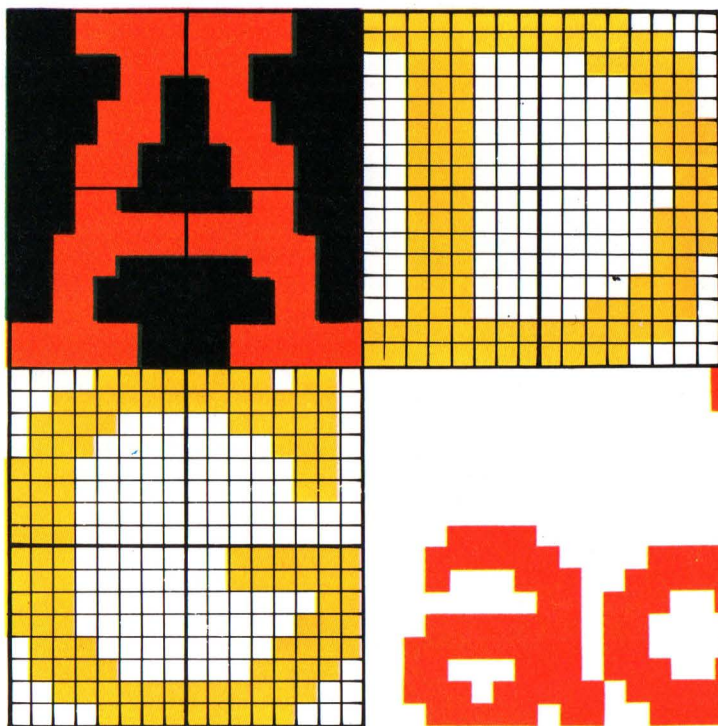
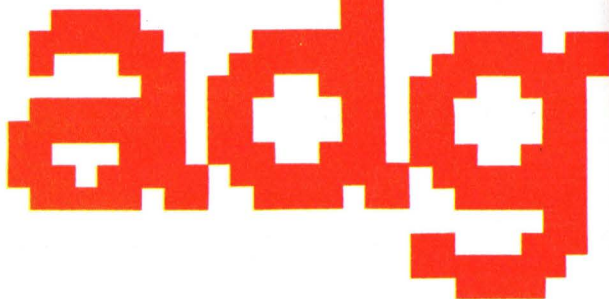
od anomalie. E questo per decine o centinaia di singole azioni.
Ancora una volta ti puoi rendere conto di quanto sia importante - prima di passare alla fase vera e propria di programmazione - valutare nei suoi più piccoli particolari tutto il complesso di operazioni, indipendentemente dal fatto che siano importanti o insignificanti che il microprocessore del robot - una volta divenuto operativo - sarà sempre chiamato a controllare.
In un robot, soprattutto se di tipo industriale, questa fase di analisi è di particolare difficoltà, visto che di solito un unico microprocessore non è in grado di affrontare da solo l'intera mole di lavoro che il robot deve poter svolgere. Molto spesso (per non dire sempre) si rende quindi necessario programmare una serie di microprocessori che sviluppino tutte le operazioni particolari (come, per esempio, muovere un braccio o afferrare un oggetto), coordinandoli attraverso un'ulteriore, grossa unità centrale di controllo, che riesca a seguire ogni momento della fase di lavorazione.
Come ben puoi immaginare, se già è difficile programmare un solo calcolatore, programmarne 10 o 15 (perché possano oltre tutto lavorare in collaborazione tra loro) diventa un'impresa estremamente delicata.

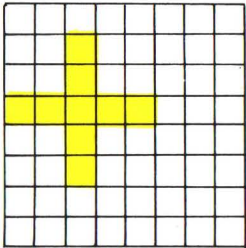
Sviluppi futuri

La robotica è una scienza molto giovane, si può dire appena agli inizi; nel suo futuro vi potranno quindi essere molti sviluppi e miglioramenti, grazie anche ai numerosi progetti di ricerca attualmente in corso in tutto il mondo.
In questo momento una delle strade più battute (a onor del vero, non solo nella robotica) è quella della cosiddetta "intelligenza artificiale".
Detta in breve, l'intelligenza artificiale sta cercando di dotare i calcolatori di una sorta di intelligenza autonoma ed automatica, capace di apprendere e ricordare gli eventi passati per

HARDWARE

HARDWARE

applicarli in quelli futuri (proprio come succede per i bambini che una volta scottati non cercano più di toccare il fuoco). Finora i risultati non sono stati molto incoraggianti, ma sembra che entro pochi anni si potrà cominciare a vedere qualcosa.
Comunque, al di là di qualsiasi timore che può incutere l'idea di macchine intelligenti, un robot capace di apprendere dall'ambiente circostante consentirebbe di risolvere molti dei problemi legati alla progettazione ed alla programmazione del robot stesso, evitando quella lunga e complessa fase di studio ed analisi che abbiamo visto essere uno degli scogli più impegnativi nella costruzione e nell'installazione di tutti i dispositivi automatici.

Anche se gli sviluppi della robotica sembrano promettere per il prossimo futuro cose molto interessanti, al momento attuale, permangono ancora diversi problemi: i robot non sanno muoversi come si vorrebbe (nessuno a tutt'oggi è riuscito a far camminare un robot come un uomo), costano abbastanza cari (soprattutto i robot industriali) e non riescono ad essere adattati per un uso universale.
In commercio ne esistono comunque già di tantissimi tipi, addirittura anche per uso personale. Lo scopo di questi ultimi, più che altro, è però rivolto all'insegnamento dei rudimenti della robotica e solo raramente consentono di ottenere prestazioni adatte per un'applicazione reale.
Essi permettono comunque di apprendere con facilità, e con una spesa relativamente limitata, il funzionamento e la programmazione dei robot, essendo molto spesso collegabili ai più diffusi personal computer per il controllo dei movimenti.

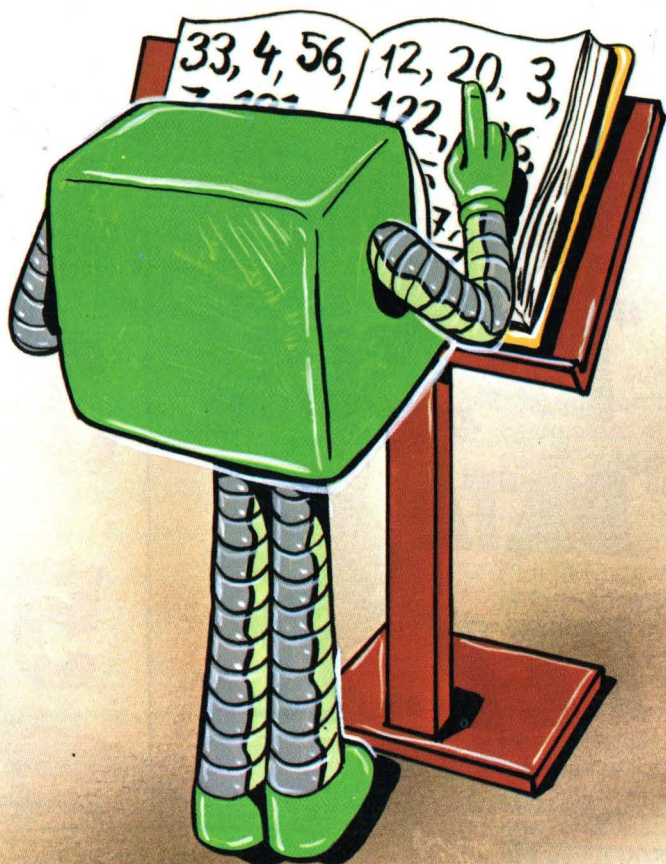
Definizione caratteri

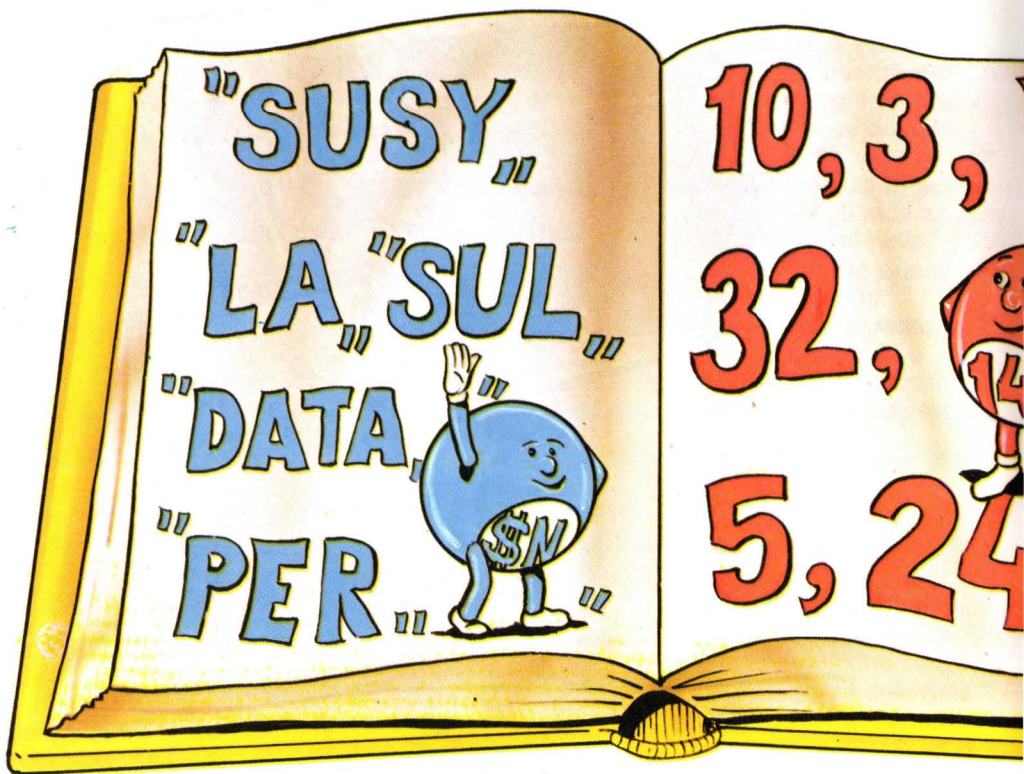
Accade talvolta che, per necessità o per divertimento, si desideri modificare uno o più caratteri tra quelli normalmente disponibili sul tuo MSX.
Per fare ciò, è necessario eseguire una semplice operazione, meglio nota come definizione dei caratteri, che consente di creare un nuovo insieme di simboli da far corrispondere a ciascuno dei tasti presenti sulla tastiera.

abcdefghij
ABCDEFG

Un carattere è racchiuso in una matrice di 8 punti per 8. Quando vuoi rappresentarlo nei dettagli è necessario operare su una matrice di punti maggiori.
Nell'esempio puoi notare come per riprodurre dettagliatamente un alfabeto particolare si sia fatto ricorso ad una matrice di 16 x 16 (4 caratteri).

Così, se per esempio volessimo poter disporre sulla tastiera lettere speciali, basterebbe che "insegnassimo" al computer la forma di ciascuno dei nuovi caratteri, in modo che ad ogni tasto corrisponda la lettera desiderata.
Ma come fare?
Ragioniamo con calma.
Poiché ogni volta che accendiamo il computer ritroviamo gli stessi caratteri, certamente questi sono memorizzati in ROM, dove non è possibile modificarli.
Per poterli vedere sullo schermo, però il sistema operativo deve certamente trasferirli nella memoria video che è RAM e può fare al caso nostro perché modificabile.
Basterà dunque conoscere l'indirizzo di inizio della tabella del profilo dei caratteri nella videoram per poter puntare ad un qualsiasi carattere e sostituirlo.
Tutto è ora predisposto per accettare le eventuali modifiche che desidereremo effettuare ai vari caratteri.
Per imparare a definire un nuovo insieme di caratteri è però necessario esaminare innanzi tutto come si deve fare per creare un singolo carattere.
Ciascun carattere è composto da una combinazione di 64 punti (ottenuta utilizzando gli 8 bit di 8 byte), disposti su 8 righe ed 8 colonne. Ecco un esempio, rappresentato da una sequenza di questo tipo:

BINARIO	DEC.	
0 0 0 0 0 0 0 0	0	
0 0 1 0 0 0 0 0	32	
0 0 1 0 0 0 0 0	32	
1 1 1 1 1 0 0 0	248	
0 0 1 0 0 0 0 0	32	
0 0 1 0 0 0 0 0	32	
0 0 0 0 0 0 0 0	0	
0 0 0 0 0 0 0 0	0	

dove ogni 1 o 0 rappresenta rispettivamente un punto (pixel) acceso o spento. Per un computer, acceso o spento significa però valore 1 o 0: basterà allora introdurre nelle locazioni adibite alla definizione di quel carattere gli otto numeri binari, letti per riga, che definiscono le varie combinazioni di pixel.
Nel nostro esempio, prendendo la seconda riga (00100000) otteniamo un certo numero binario, che, convertito in decimale (32), ci fornisce uno degli otto valori da inserire nelle locazioni della memoria che specificheranno la descrizione e la struttura di quel carattere.
Per chiarezza questo valore è stato scritto sulla destra della medesima riga.
La stessa operazione sulle altre sette righe ci procura infine i restanti valori, necessari per definire completamente tutto il carattere.
Gli otto numeri che derivano da ⊞ sono pertanto: 0, 32, 32, 248, 32, 32, 0 e 0. A questo punto li si potrà inserire nella memoria.
Quando il computer dovrà visualizzare questo carattere preleverà allora gli otto numeri dalla memoria e li visualizzerà sullo schermo. Questo è tutto.
Il primo passo da eseguire nella definizione di un carattere è perciò quello di disegnare una serie di 64 quadretti nei quali definire i pixel da accendere o da spegnere.
Creando un qualsiasi carattere è buona norma cercare di non usare mai uno 0 od un 1 isolato: infatti, adoperando un televisore di qualità non ottima, è probabile che il punto corrispondente non venga visualizzato.
Fatta la prima volta, la via da seguire per impostare altri caratteri è sempre la stessa: cambieranno soltanto i numeri da inserire nel computer (ogni carattere è chiaramente definito da una particolare ed unica sequenza di punti luminosi) e gli indirizzi delle locazioni nelle quali introdurre questi numeri. Alla fine l'insieme dei caratteri sarà stato definito e potrà essere utilizzato a piacimento da qualsiasi programma.
Per recuperare il vecchio insieme di caratteri sarà sufficiente spegnere il computer, riaccendendolo dopo qualche secondo: questa operazione cancellerà tutte le modifiche effettuate nella memoria RAM, ripristinando gli usuali valori.

READ/DATA

Le istruzioni READ e DATA permettono di leggere dei dati all'interno di un programma, evitando così di doverli impostare manualmente dalla tastiera.
L'introduzione dei dati da elaborare avviene inoltre senza il momentaneo arresto del programma, al contrario di quanto accade per l'istruzione INPUT.
Con ogni probabilità ti starai però chiedendo quando mai potrà servirti una simile struttura, visto che finora sei riuscito benissimo a farne a meno.
Un esempio ti chiarirà subito le idee.

LINGUAGGIO

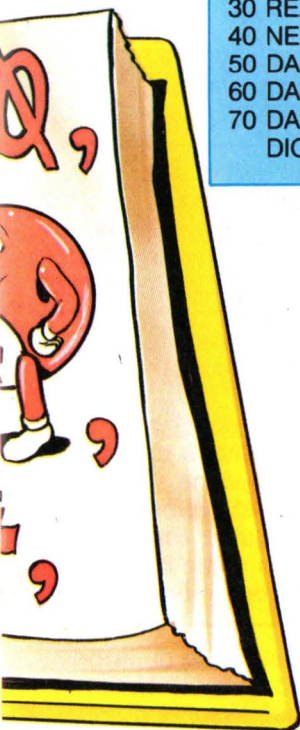
Accade molto spesso che un programma richieda come azione preliminare dell'esecuzione la cosiddetta inizializzazione delle variabili, cioè l'inserimento di alcuni specifici valori in determinate variabili (per esempio i nomi e le durate dei singoli mesi dell'anno).
Per fare questa operazione si hanno a disposizione tre possibili alternative:
— utilizzare una lunga serie di LET all'inizio del programma;
— richiedere ad ogni esecuzione, utilizzando delle INPUT, i valori da assegnare alle singole variabili;
— adoperare READ e DATA.
Scartiamo subito la prima soluzione: richiederebbe troppo lavoro durante la battitura del programma e farebbe occupare troppa memoria (ricorda che ogni istruzione conservata nel computer occupa una certa porzione di memoria).
La seconda soluzione è già più accettabile. Nel

caso dei mesi si potrebbe scrivere:

```
10 DIM M$ (12), G (12)
20 FOR I = 1 TO 12
30 INPUT M$ (I), G (I)
40 NEXT I
```

e con queste poche istruzioni ce la saremmo cavata.

Il problema è però risolto solo parzialmente: ad ogni RUN dovremmo infatti metterci di buona lena a battere GENNAIO, 31, FEBBRAIO, 28, ... ecc., impostando cioè dei valori che, tutto sommato, tra un'esecuzione e l'altra non subiscono alcuna modifica e che quindi sarebbe comodo conservare in permanenza nel programma. Ciò significa che ogni volta che utilizziamo questo programma dobbiamo assegnare da tastiera tutte le 24 informazioni che ci vengono richieste. Lavoro lungo, noioso ed inutile.
Adottando READ e DATA avremo invece:

```
10 DIM M$ (12), G (12)
20 FOR I = 1 TO 12
30 READ M$ (I), G (I)
40 NEXT I
50 DATA GENNAIO, 31, FEBBRAIO, 28, MARZO, 31, APRILE, 30
60 DATA MAGGIO, 31, GIUGNO, 30, LUGLIO, 31, AGOSTO, 31
70 DATA SETTEMBRE, 30, OTTOBRE, 31, NOVEMBRE, 30,
   DICEMBRE, 31
```

In pratica queste istruzioni ordinano al calcolatore di fare le stesse cose delle linee viste prima, con la sola differenza che le singole variabili degli array M$ ( ) e G ( ) devono essere lette (READ) non più dalla tastiera - come accadeva prima a causa dell'INPUT - ma dalle righe DATA.
I dati vengono quindi inseriti nel calcolatore una sola volta e, cosa ancora più importante, chi usa il programma

non è tenuto a conoscere ed a battere questi dati (prova a pensare ai nomi di tutte le squadre del campionato di calcio o ad altre serie di dati, magari meno noti dei nomi dei mesi dell'anno).
In questo modo è possibile mantenere - conservati all'interno del programma - dei valori che altrimenti andrebbero tutte le volte dispersi con lo spegnimento del calcolatore o con la riesecuzione del programma stesso.
Le istruzioni DATA contengono i diversi valori che si vogliono assegnare. Esse permettono di riunire tutti i dati in un unico punto del programma, di solito all'inizio o alla fine, dove sono più leggibili.
L'istruzione READ consente di leggere i dati che sono memorizzati nelle DATA.
È ovvio che si debba fare estrema attenzione affinché il tipo della variabile dell'istruzione READ corrisponda al contenuto dell'istruzione DATA, proprio come succede negli INPUT da tastiera: non si dovrà mai verificare che una variabile numerica possa contenere un valore alfanumerico o che in una variabile di tipo stringa si trovi un valore numerico. Questo tipo di errore è sempre in agguato!

Le linee DATA del listato precedente sono tre solo per motivi di leggibilità: avremmo infatti potuto metterne un numero diverso anche non necessariamente consecutive.
Infatti se vi sono più linee DATA, in punti diversi del programma, queste sono lette di seguito, come se si trattasse di un'unica linea.

Esempi

```
10 READ A, B
20 DATA 3, 5
```

Alle variabili A e B vengono assegnati i valori 3 e 5.

```
10 READ A, B, C
20 DATA 3, 5
```

Non ci siamo: alla variabile C non corrisponde alcun valore e si avrà il messaggio di errore OUT OF DATA in 20.

```
 5 READ A, A$, C
10 DATA 3, 4, 5
```

A, A$ e C assumono rispettivamente valore 3, "4" e 5.
Poiché la seconda variabile (A$) è di tipo stringa, 4 viene considerato valore stringa quindi inserito in memoria come carattere e non come numero.

```
 5 READ C$, A, D$
10 DATA PIPPO, PLUTO
```

In questo caso gli errori sono due: la variabile A non può assumere

valore PLUTO (essendo di tipo numerico) e la variabile D$ è sprovvista del corrispondente DATA.

```
 5 FOR I = 1 TO 6
 8 READ K$
10 PRINT K$
13 NEXT I
20 DATA FRANCO, MARIO
30 DATA MATTEO, CARLO
40 DATA PIPPO, GIACOMO
```

La variabile K$ assumerà via via i 5 valori specificati nelle istruzioni DATA. Il comando PRINT K$, inserito nel ciclo FOR, provocherà pertanto la stampa di FRANCO, MARIO, ..., GIACOMO. Nota come la variabile K$ sia dello stesso tipo (stringa) dei valori che dovrà assumere in seguito alle READ.

```
 5 CLS
10 PRINT "SISTEMA SOLARE" : PRINT
15 FOR K = 1 TO 9
20 PRINT
25 READ P$, D
30 PRINT "IL PIANETA "; P$
35 PRINT "HA DIAMETRO DI"; D; "KM"
40 NEXT K
45 DATA MERCURIO, 4880, VENERE, 12096
50 DATA TERRA, 12740, MARTE, 6780
55 DATA GIOVE, 141560, SATURNO, 120800
60 DATA URANO, 51000, NETTUNO, 49300
65 DATA PLUTONE, 7000
```

Questo, più che un esempio, è una dimostrazione dell'utilità di READ ... DATA. Con poche istruzioni è infatti possibile inserire in permanenza delle informazioni che non cambieranno mai tra un'esecuzione e l'altra, consentendo un considerevole risparmio sia di spazio che di tempo.

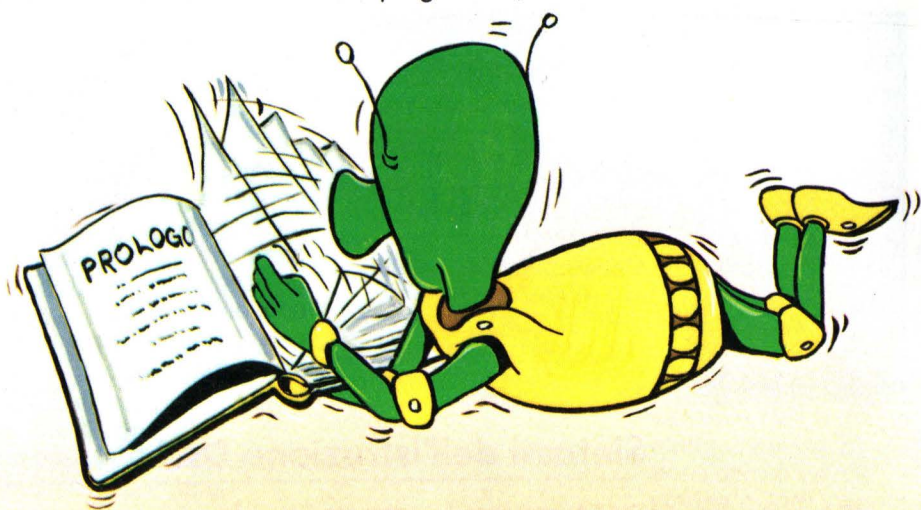
Sintassi dell'istruzione DATA

DATA costante [, costante] [, ...]

Sintassi dell'istruzione READ

READ variabile [, variabile] [, ...]

RESTORE

Per controllare l'ordine secondo cui vengono prelevati i valori delle DATA possiamo immaginare che all'interno della memoria esista una specie di indice (chiamato anche puntatore) che segnali - di volta in volta - qual è il successivo valore che può essere letto con un'istruzione READ.
Ogni volta che il computer legge un qualsiasi elemento dalle linee DATA questo puntatore viene perciò fatto spostare sull'elemento successivo, in modo che l'interprete BASIC sappia sempre sin dove è arrivata la lettura dei vari termini delle DATA.
Ad ogni READ corrisponderà quindi un automatico avanzamento del puntatore delle DATA.
Talvolta può comunque essere molto utile avere la possibilità, nel corso del programma, di accedere alle stesse informazioni più di una volta.
L'istruzione RESTORE serve appunto per riportare il puntatore interno alla prima linea DATA del programma rendendo nuovamente disponibili le costanti come se non fossero mai state lette.
Nel caso invece, il valore da rileggere non fosse il primo, è necessario indicare dopo RESTORE il numero della linea data da cui iniziare a prelevare gli elementi.

```
10 DATA DO, RE, MI, FA, SOL
20 DATA GIALLO, VERDE
30 FOR I = 1 TO 7 : READ A$
40 PRINT A$ : NEXT I
```

Il programma fino a questo punto stampa tutti gli elementi delle linee DATA.
Cioè le note e i colori.
Il puntatore è posizionato dopo la costante "verde".
Ecco come proseguire per ristampare solamente i colori:

```
50 RESTORE 20: REM POSIZIONE DEL
   PUNTATORE SUL "GIALLO" DELLA LINEA 20
60 FOR I = 1 TO 2 : READ A$
70 PRINT A$ : NEXT I
```

Le linee 60 e 70 provvedono alla ristampa dei colori "GIALLO" e "VERDE"
Quando viene incontrata, l'istruzione RESTORE - indipendentemente dal numero di elementi che sono già stati letti - fa quindi in modo che la successiva istruzione READ torni indietro a leggere il primo elemento nella lista dei DATA, proprio come accade la prima volta.
Così, il seguente programma

```
10 FOR I = 1 TO 1000
20 READ A
25 PRINT A
30 RESTORE
40 NEXT I
50 DATA 27, 10, 60
```

avrà come unico risultato la stampa per 1000 volte del valore 27, visto che il RESTORE posto alla linea 20 impedisce al puntatore delle DATA di avanzare al termine successivo.
Prova adesso a togliere la riga 30 ed a eseguire il programma: se avevi dei dubbi sul funzionamento di RESTORE, li chiarirai subito.

Sintassi dell'istruzione

RESTORE [NUMERO LINEA DATA]

Gli Sprite

Una delle caratteristiche più interessanti dei sistemi MSX consiste nella disponibilità di elementi grafici, detti sprite, che possono essere definiti e visualizzati con estrema facilità.

Uno sprite non è altro che un piccolo disegno contenuto in un quadratino di piccole dimensioni che si può far comparire in qualsiasi punto dello schermo senza interferire con quello che precedentemente era visualizzato. Gli Sprite esistono in quattro differenti formati, disponibili uno alla volta. Per scegliere il formato si usa l'istruzione SCREEN; dopo il tipo di schermo da usare va specificato uno di questi numeri:

0 - Formato 8x8
1 - Formato 8x8 ingrandito
2 - Formato 16x16
3 - Formato 16x16 ingrandito.

```
SCREEN 1,1
```

predispone il formato 8x8 ingrandito per il modo testo 24x32. Possono essere memorizzati fino a 256 differenti disegni di sprite nel formato 8x8, mentre tale numero è ridotto a 64 nel formato 16x16.
Non si possono visualizzare più di 32 sprite contemporaneamente; inoltre su una stessa linea orizzontale non si possono trovare più di 4 sprite, altrimenti il quinto non verrebbe visualizzato. Il modo SCREEN 0 (24x40) non consente l'utilizzo degli sprite.

La funzione SPRITE$

Per definire il disegno di uno sprite bisogna immaginarlo disegnato su un foglio di carta trasparente a quadretti. Ogni quadretto può essere lasciato trasparente oppure riempito di inchiostro colorato.

Se si fa corrispondere uno zero ai quadretti vuoti ed uno a quelli pieni, si ottengono dei numeri in formato binario.
La seguente sequenza di numeri definisce uno sprite a forma di freccia rivolta verso l'alto.

```
&B00010000  =16
&B00111000  =56
&B01111100  =124
&B11111110  =254
&B00010000  =16
&B00010000  =16
&B00010000  =16
&B00010000  =16
```

Ad ognuno dei numeri così trovati si fa corrispondere il relativo carattere, e con i caratteri così ottenuti si forma una stringa.
Il primo carattere definisce la riga più in alto; l'ultimo la riga più in basso.
Se si assegna questa stringa alla funzione STRING$( ) con un numero compreso tra 0 e 255 come parametro, si è definito uno sprite, che potrà essere in seguito utilizzato riferendosi a quel numero.

```
10 SCREEN1,0 : REM SELEZIONA IL MODO TESTO 24x32 E IL FORMATO
   SPRITE 8x8
20 SPRITE$(1)=CHR$(16)+CHR$(56)+CHR$(124)+$(254)+CHR$(16)+
   CHR$(16)+CHR$(16)+CHR$(16) : REM 8 BYTE PER DEFINIRE LO SPRITE
   N. 1 A FORMA DI FRECCIA
```

Se si usano sprite di formato 16x16, li si dovrà considerare come formati da 4 sprite 8x8 disposti come segue:

1	3
2	4

La stringa di definizione ha in questo caso 32 caratteri, e si ottiene concatenando le 4 stringhe di 8 caratteri che definiscono gli sprite più piccoli.

Esempi

SPRITE$(1)=STRING$ (8,255)	Definisce lo sprite numero 1 come costituito da 8 righe uguali di tutti 1 (255=&B11111111), formanti un quadrato pieno.
SPRITE$(0)=A$	Definisce il disegno dello sprite numero 0 nel modo indicato dai caratteri della stringa A$, che deve essere definita in precedenza.

L'istruzione PUT SPRITE

Con questa istruzione puoi fare apparire uno sprite in qualsiasi punto dello schermo, con il colore che desideri. Immagina che davanti allo schermo ci siano 32 fogli trasparenti, uno sopra l'altro.

Su ciascuno di questi fogli puoi fare apparire uno sprite, il quale nasconderà temporaneamente quello che gli sta dietro.
I fogli trasparenti sono denominati "PIANI DEGLI SPRITE", e sono individuati da un numero compreso da 0 a 31.
Sui ciascun piano non si può visualizzare più di uno sprite contemporaneamente; quando se ne disegna uno, il precedente sparisce.
La posizione in cui lo sprite viene visualizzato va indicata tramite due coordinate che individuano il pixel in alto a sinistra dello sprite.
Gli altri dati da indicare sono il colore dello sprite, secondo il codice numerico solito, e il numero che individua il disegno dello sprite, lo stesso usato nella funzione SPRITE$.

Esempi

PUTSPRITE 0,(00), 1, 2	Visualizza lo sprite con disegno numero 2 sul piano numero 0 (il più avanzato) alle coordinate (0,0) (in alto a sinistra) col colore 1 (nero).

```
PUTSPRITE1,(250, 180),
15, 9
```

Visualizza lo sprite con disegno numero 9 sul piano numero 1 alle coordinate (250,180) (in basso a destra) col colore 15 (bianco).
Riprendiamo ora il programma in cui abbiamo definito uno sprite a forma di freccia e vediamo, grazie a PUT SPRITE, di farla apparire.

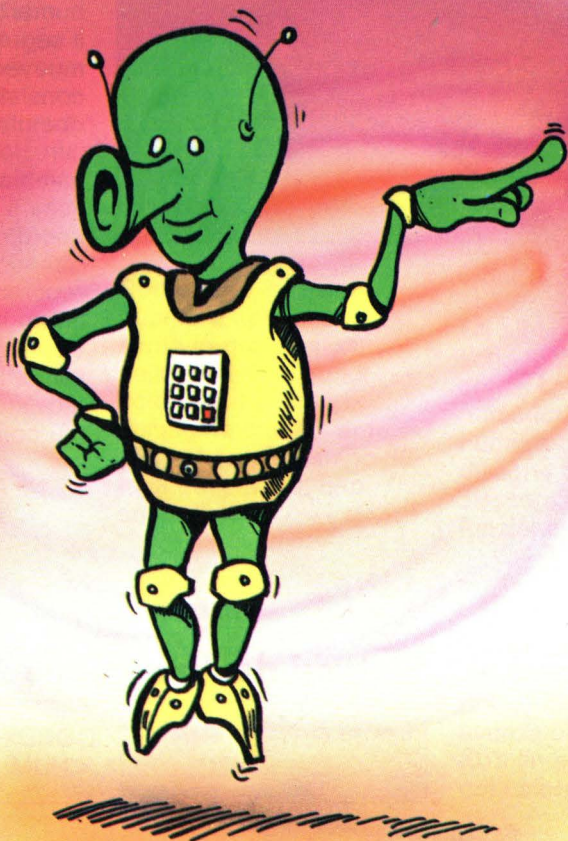
```
10 SCREEN1,1 : REM SELEZIONA IL MODO
   TESTO 24x32 E IL FORMATO SPRITE 8x8
   INGRANDITO
20 SPRITE$(1)=CHR$(16)+CHR$(56)+CHR$
   (124)+CHR$(254)+CHR$(16)+CHR$
   (16)+CHR$(16)+CHR$(16) : REM 8 BYTE PER
   DEFINIRE LO SPRITE N. 1
30 PUT SPRITE 1,(32,8), 1, 1 : REM PER
   VISUALIZZARE LO SPRITE
```

Già che ci siamo, però attiviamo il formato ingrandito.
Per vedere la differenza tra i formati degli sprite cambia la linea 10 in SCREEN1,0.
Per fare scomparire uno sprite visualizzato, basta assegnare alla coordinata Y il valore 209; usando invece il valore 208 si fanno sparire, oltre allo sprite interessato, tutti gli sprite visualizzati sui piani di numero superiore.
Il segreto per far muovere uno sprite consiste nel visualizzarlo ripetutamente con varie istruzioni PUT SPRITE cambiando di volta in volta il valore delle coordinate.
Usando vari disegni che rappresentano una figura in pose diverse e visualizzandoli uno dopo l'altro, l'effetto del movimento sarà ancora più completo.

Sintassi dell'istruzione

PUT SPRITE<PIANO>, <COORDX>, <COORDY>, <COLORE>, <NUM. SPRITE>

Definizione di un carattere

Tu sai già come il tuo MSX possa visualizzare tutti i caratteri che puoi vedere sul video, e sai anche che le immagini di questi caratteri (una serie di 8 byte per ogni carattere) sono gelosamente custodite in ROM (memoria a sola lettura), perchè non vengano perse ogni volta che spegni il calcolatore. Può sembrare, quindi che ogni volta che verrà premuto il tasto "A" del MSX il video mostrerà una A. Invece il tuo MSX ti dà anche la possibilità di creare uno per uno tutti i caratteri. Proviamo a definire un carattere che assomigli a un omino! Usando la stessa tecnica, con cui sono

disegnati normalmente i caratteri, costruiamo una matrice 8 per 8 e riempiamola in modo da ottenere il simbolo voluto, usando 0 per i punti spenti e 1 per quelli accesi.

POTENZE DI 2

128	64	32	16	8	4	2	1	ADDIZIONE	VALORE DECIMALE DA "POKARE"
0	0	0	1	1	0	0	0	→ 16 + 8	= 24
0	0	0	1	1	0	0	0	→ 16 + 8	= 24
0	0	1	1	1	1	0	0	→ 32 + 16 + 8 + 4	= 60
0	1	0	1	1	0	1	0	→ 64 + 16 + 8 + 2	= 90
1	0	0	1	1	0	0	1	→ 128 + 16 + 8 + 1	= 153
0	0	1	0	0	1	0	0	→ 32 + 4	= 36
0	0	1	0	0	1	0	0	→ 32 + 4	= 36
0	1	1	0	0	1	1	0	→ 64 + 32 + 4 + 2	= 102

Calcolo dei valori necessari per definire il nuovo carattere.

Ora ci tocca fare un pò di conti: la prima riga (cioè il primo byte del carattere) contiene il numero binario 00011000, cioè 24 decimale. Facendo lo stesso calcolo per le altre righe otteniamo una serie di 8 numeri cioè 24, 24, 60, 90, 153, 36, 36 e 102.
Poichè come sai non è possibile modificare un carattere nella ROM è necessario intervenire sui caratteri da sostituire in VIDEORAM .
Occorre quindi conoscere da quale locazione della VIDEORAM sono presenti i byte che compongono, a gruppi di otto, i vari caratteri.

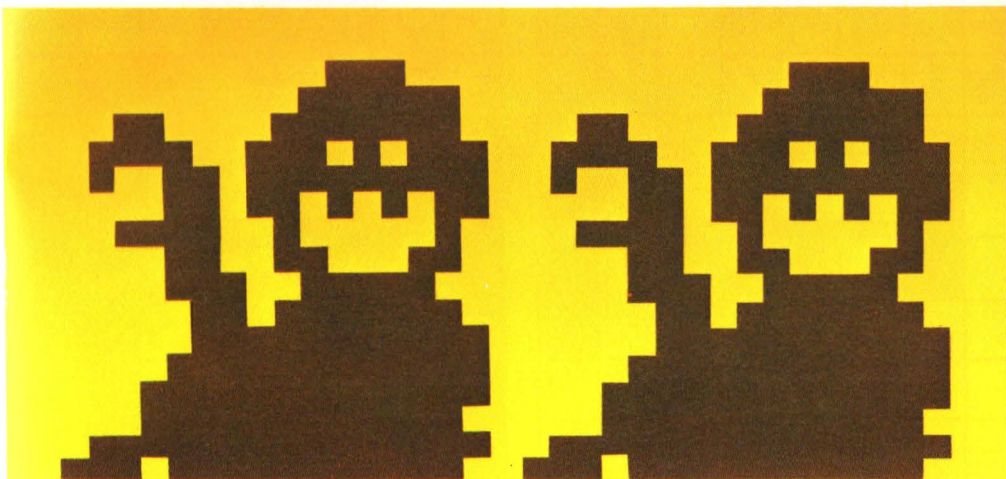
Poiché questo indirizzo varia col variare del modo di schermo, facciamo ricorso ad una apposita funzione "base" in grado di restituire l'indirizzo cercato.
Base ha unicamente bisogno di un argomento che le indichi cosa cercare.

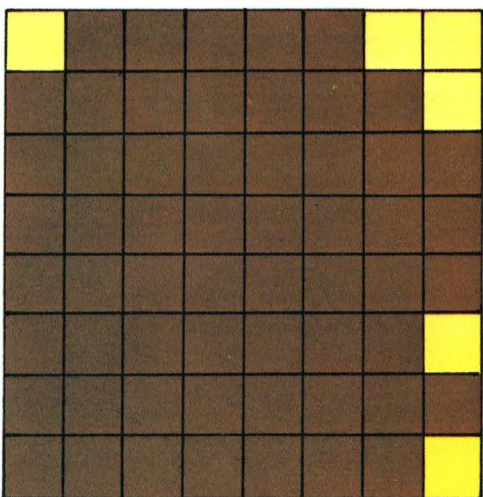
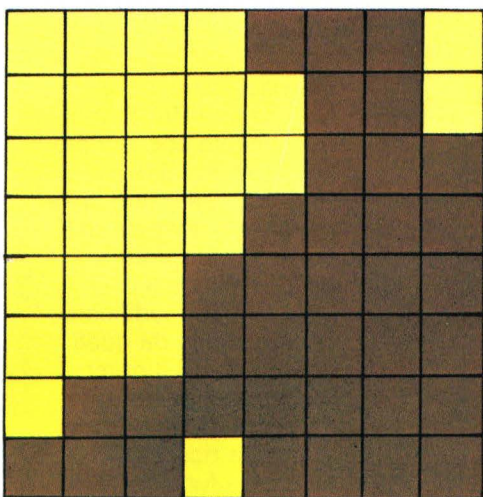
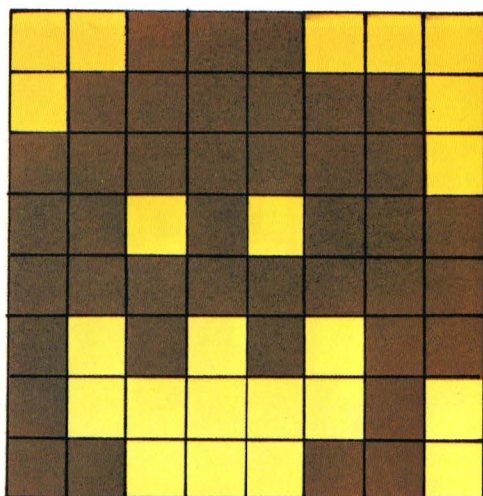
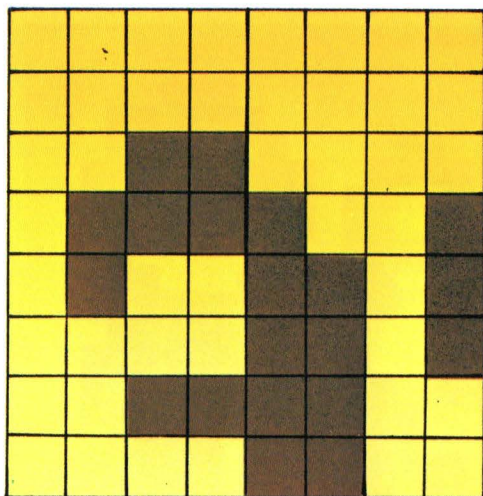
Con argomento (2) restituisce l'indirizzo di partenza della tabella del set dei caratteri in SCREEN 0. Con argomento (7) quella in SCREEN 1.
Queste "stranezze" sono presenti nel listato che segue e potrai farne uso ogni volta che desideri disegnare e assegnare nuovi caratteri.
Obiettivo del programma è sostituire un qualsiasi carattere digitale della tastiera con quello dell'omino progettato in precedenza.

```
10 SCREEN1:WIDTH30:D=BASE(7)
20 INPUT "quale carattere vuoi modificare"; C$
30 IC=D+8*ASC(C$)
40 FORI=IC TO IC+7
50 READ A : VPOKEI,A
60 NEXT
70 END
80 DATA 24, 24, 60, 90, 153, 36, 36, 102
```

Cambiando la linea 10 del listato puoi visualizzare l'omino anche in SCREEN0, cosa impossibile, invece, con uno sprite.

```
10 SCREEN0 : WIDTH30 : D=BASE(2)
```

Poichè in questo modo schermo vengono prese in considerazione solo le prime sei colonne verticali di pixel di un carattere, il nostro eroe apparirà non completamente.

Commento al listato

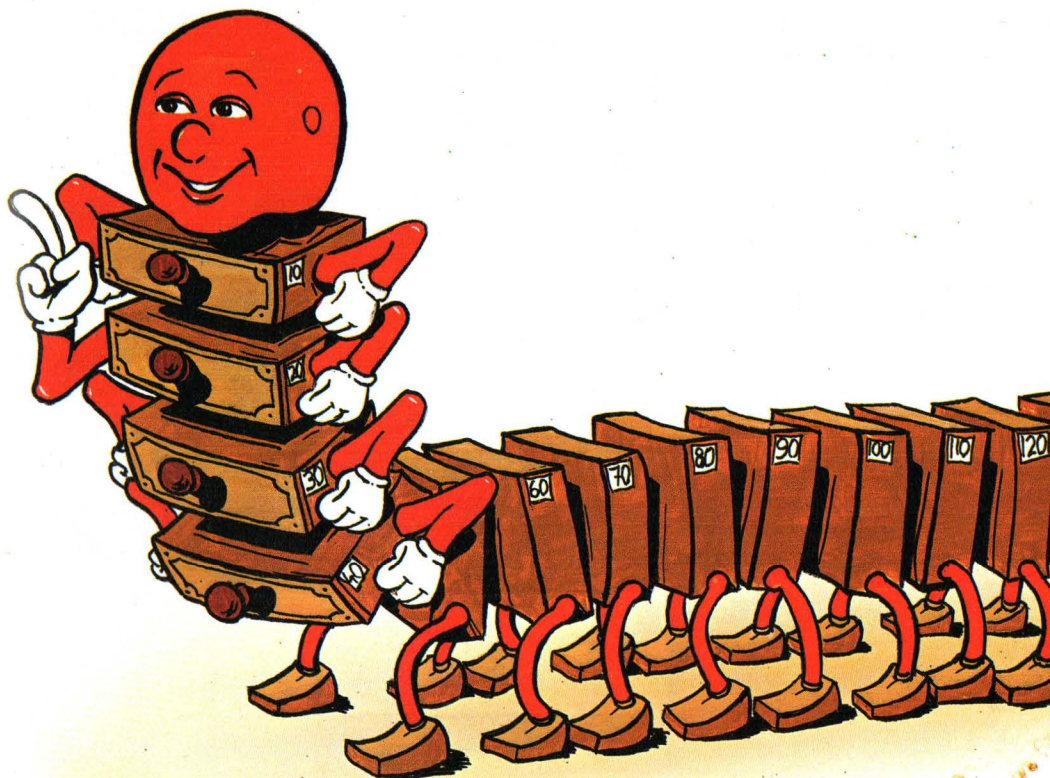
La linea 10 seleziona il modo di schermo ed il numero di colonne su cui visualizzare il testo.
La funzione BASE con l'argomento appropriato ha il compito di fornire l'indirizzo del primo byte del primo carattere del set nella videoram.
La 20 richiede il carattere da modificare.
La 30 calcola da quali indirizzi parte il primo degli otto byte del carattere da modificare.
In questa ricerca si utilizza la funzione ASC che fornisce il codice ASCII del carattere scelto.
Indirizzo Inizio Carattere (IC)=codice ASCII del carattere moltiplicato 8+Indirizzo Inizio Tabella (D).
La 40 apre il ciclo che per otto volte..

Definire i caratteri vi permette di visualizzare anche immagini di fantasia.

La 50...legge i dati dell'omino e li scrive nella videoram (VPOKE) proprio sopra a quelli del carattere scelto, sostituendolo.
Linea 60 fine ciclo.
Linea 70 fine programma.
Linea 80 dati decimali del carattere "omino".
Adesso ogni volta che premi il tasto prescelto, vedrai apparire il nuovo personaggio.
Nota bene: poichè non sono stati trasferiti i caratteri in reverse il cursore può assumere una strana forma.

L'errore è sempre in attesa: malignamente aspetta una tua distrazione, anche minima.
Non dargli mai la possibilità di sogghignare.
Impara la sintassi delle istruzioni.
Realizza programmi in forma modulare, facilmente leggibili, e molto documentati.
Nella digitazione, poi, metti la massima attenzione: anche una virgola al posto sbagliato, compromette la buona riuscita del programma.

Indicativo presente verbi in ARE

Non è facile, ma certo divertente, insegnare ad un computer la coniugazione di un verbo.
Proviamoci con questo programma che ha lo scopo di mettere in grado il tuo MSX di elaborare e visualizzare l'indicativo presente dei verbi regolari transitivi della I[a] coniugazione (quelli dell'infinito in ARE).

```
10 DATA IO, O, TU, I, EGLI, A
20 DATA NOI, IAMO, VOI, ATE, ESSI, ANO
30 CLS: PRINT"INSERISCI CAPS LOCK" : INPUT"INFINITO"; V$
40 R$=RIGHT$(V$, 3)
50 IFR$<>"ARE" THEN30
60 N=LEN(V$) : C$=MID$(V$, N-3, 1)
70 PRINT : PRINT
80 FORA=1TO6
90 READA$, B$
100 R$=LEFT$(V$, N-3)
110 T$=LEFT$(B$, 1)
120 IF C$=T$ THENR$=LEFT$(V$, N-4)
130 PRINTA$; TAB(5); R$; B$
140 NEXT
150 PRINT : PRINT : PRINT "ANCORA? S/N"
160 A$=INKEY$ : IFA$=" "THEN160
170 IFA$="S"THEN RUN
180END
```

VIDEOESERCIZI

Quali parole (costante stringa) verranno stampate?

```
 10 READ A
 20 READ B
 30 FOR I = 1 TO A * B
 40 READ A$
 50 NEXT I
 60 READ A$
 70 PRINT A$
 80 END
 90 DATA 3, 1
100 DATA "VIDEO"
110 DATA "BASIC"
120 DATA "JACKSON"
130 DATA "SOFTIDEA"
140 DATA "COMMODORE"
150 DATA "MSX"
160 DATA "IBM"
170 DATA "CHIP"
180 DATA "6502"
190 DATA "Z80"
200 DATA "VELOCE"
```

Molti discorsi di uomini politici sembrano avere la struttura impostata da questo programma cambiano opportunamente le linee data con vocabili come "congiuntura", "piattaforma" ecc.. ne otterrai degli esempi.

```
 10 RESTORE 140
 20 A=INT(RND(—TIME)*9)+1
 30 FOR K=1TOA
 40 READ A$ : NEXT K
 50 A=INT(RND(—TIME)*9)+1
 60 RESTORE 160
 70 FOR K=1TO A
 80 READ B$ : NEXT K
 90 CLS : PRINTA$; " "; B$
100 PRINT : PRINT "ANCORA? (S/N)"
110 Z$=INKEY$ : IFZ$=" " THEN110
120 IFZ$="S" THENRUN
130 END
140 DATA LA GIRAFFA, L'UOMO, IL CANE, LA MUCCA, L'ASINO,
    IL CAVALLO
150 DATA L'ELEFANTE, IL GATTO, LA GALLINA, IL CONIGLIO
160 DATA SGHIGNAZZA, SCOPPIA, SALTICCHIA, SVOLAZZA,
    RIMBALZA, FUGGE
170 DATA NITRISCE, BARRISCE, MUGGISCE, SORRIDE
```